Logo

Description automatically generated

**DOCUMENTAȚIE:**

**MIPS 16**

STUDENT: SABĂU OANA-MARIA

GRUPA: 30222

AN UNIVERSITAR: 2022/2023

**Cuprins**

[1. Obiectiv 3](#_Toc133751569)

[2. Prezentare și analiză problemă 3](#_Toc133751570)

[3. Definire instrucțiuni și trasare program 4](#_Toc133751571)

[4. Implementare 5](#_Toc133751572)

[5. Scheme incluse 6](#_Toc133751573)

[6. Concluzii 12](#_Toc133751574)

[7. Resurse necesare 12](#_Toc133751575)

[8. Bibliografie 12](#_Toc133751576)

# **1. Obiectiv**

Obiectivul proiectului constă în realizarea unui microprocesor MIPS ciclu unic pe 16 biți care creează un model digital al unui procesor care poate executa instrucțiuni MIPS la viteze ridicate, folosind limbajul de descriere hardware VHDL. Acest procesor poate fi implementat pe un chip de siliciu sau pe o placă FPGA și poate fi programat cu cod mașină pentru a rula aplicații specifice MIPS. În general, obiectivul este de a crea un procesor personalizat care poate fi utilizat într-o varietate de aplicații, de la routere și switch-uri la playere de jocuri și alte dispozitive de calcul, iar în acest caz va executa un program ales de mine.

# **2. Prezentare și analiză problemă**

MIPS, un acronim pentru **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages, a fost un proiect de cercetare condus de John L. Hennessy la Universitatea Stanford între 1981 și 1984. MIPS a investigat un tip de Instruction Set Architecture (ISA), numită în prezent Reduced Instruction Set Computer (RISC), implementarea sa ca microprocesor fiind realizată prin intermediul tehnologiei semiconductoarelor cu integrare la scară foarte mare (VLSI) și exploatarea eficientă a arhitecturilor RISC cu ajutorul compilatoarelor de optimizare.

Microprocesorul studiat la cursul de Arhitectura Calculatoarelor este MIPS 32. Microprocesorul studiat și implementat la laborator va fi o versiune simplificată a procesorului descris la curs, adică MIPS 16. Setul de instrucțiuni va fi mai mic, dimensiunea instrucțiunilor/a cuvântului va fi pe 16 biți, și, implicit, vor fi un număr redus de registre de uz general, respectiv o dimensiune mai mică pentru memorie. Principiile de proiectare din curs rămân valabile (calea de date, control). Principalul motiv pentru simplificarea pe 16 biți este dat de modalitățile restrânse de afișare de pe placa de dezvoltare (leduri, afișorul SSD) și se facilitează procesul de trasare / testare a programului exemplu pe procesorul implementat.

Programul ales pentru a fi implementat de microprocesor determină cel mai mare divizor comun – CMMDC (eng. GCD) dintre două elemente dintr-un array a căror poziții au fost date. Programul inițial este scris în limbajul de programare C, apoi este transformat în cod de asamblare, iar apoi în cod mașină (respectând convențiile MIPS 16).

# **3. Definire instrucțiuni și trasare program**

*Transformarea programului în cod mașină*

*Program C : CMMDC dintre două elemente dintr-un array a căror poziții au fost date*

#include <stdlib.h>

#include <stdio.h>

int main() {

int v[] = { 5, 12, 4, -3, 17, 32, 8 };

int a = 1;

int b = 4;

while (v[a] != v[b]) {

if (v[a] > v[b]) {

v[a] = v[a] - v[b];

}

else {

v[b] = v[b] - v[a];

}

}

//rezultatul este stocat fie in v[a], fie in v[b]

return 0;

}

*Program MIPS asamblare:*

**0 LW $1, 1($0)**

**1 LW $2, 5($0)**

**2 BEQ $1, $2, 6**

**3 SLT $3, $1, $2**

**4 BEQ $3, $0, 2**

**5 SUB $2, $2, $1**

**6 J 2**

**7 SUB $1, $1, $2**

**8 J 2**

**9 SW $1, 7($0)**

*Program cod mașină:*

**B"010\_000\_001\_0000001", -- X"4081" -- lw $1, 1($0) --0**

**B"010\_000\_010\_0000101", -- X"4105" -- lw $2, 5($0) --1**

**B"100\_010\_001\_0000110", -- X"8886" -- beq $1, $2, 6 --2**

**B"000\_001\_010\_011\_0\_111", -- X"0537" -- slt $3, $1, $2 --3**

**B"100\_000\_011\_0000010", -- X"8182" -- beq $3, $0, 2 --4**

**B"000\_010\_001\_010\_0\_001", -- X"08A1" -- sub $2, $2, $1 --5**

**B"111\_0000000000010", -- X"E002" -- j 2 --6**

**B"000\_001\_010\_001\_0\_001", -- X"0511" -- sub $1, $1, $2 --7**

**B"111\_0000000000010", -- X"E002" -- j 2 --8**

**B"011\_000\_001\_0000111", -- X"6087" --sw $1, 7($0) --9**

# **4. Implementare**

Ciclul de execuție a unei instrucțiuni MIPS are următoarele etape / faze:

* ***IF – Extragerea Instrucțiunii / Instruction Fetch*** - procesorul extrage instrucțiunea din memoria programului. Adresa instrucțiunii este stocată în registrul Program Counter (PC), iar PC-ul este incrementat pentru a indica adresa următoarei instrucțiuni din program.
* ***ID/OF – Decodificarea Instrucțiunii / Extragerea Operanzilor*** - acum, procesorul decodează instrucțiunea extrasă în faza IF și extrage orice operanzi necesari pentru execuție. În funcție de tipul instrucțiunii, se poate lua decizia de a transfera datele din registrele de intrare ale procesorului în unitatea de execuție (EX), sau de a citi datele din memoria sistemului.
* ***Instruction Decode / Operand Fetch*** - în această fază, procesorul efectuează operația specifică instrucțiunii pe operanzii decodați în faza ID/OF
* ***EX – Execuție / Execute*** - procesorul accesează memoria sistemului pentru a citi sau scrie datele. Această fază este necesară doar pentru instrucțiunile care accesează memoria.
* ***MEM – Memorie / Memory*** - procesorul accesează memoria sistemului pentru a citi sau scrie datele. Precum faza EX, este necesară doar pentru instrucțiunile care accesează memoria.
* ***WB – Scriere Rezultat / Write Back*** - procesorul scrie rezultatul operației înapoi în registrele de ieșire ale procesorului. Acest lucru este necesar pentru a păstra consistența datelor și pentru a permite instrucțiunilor ulterioare să utilizeze valorile corecte.

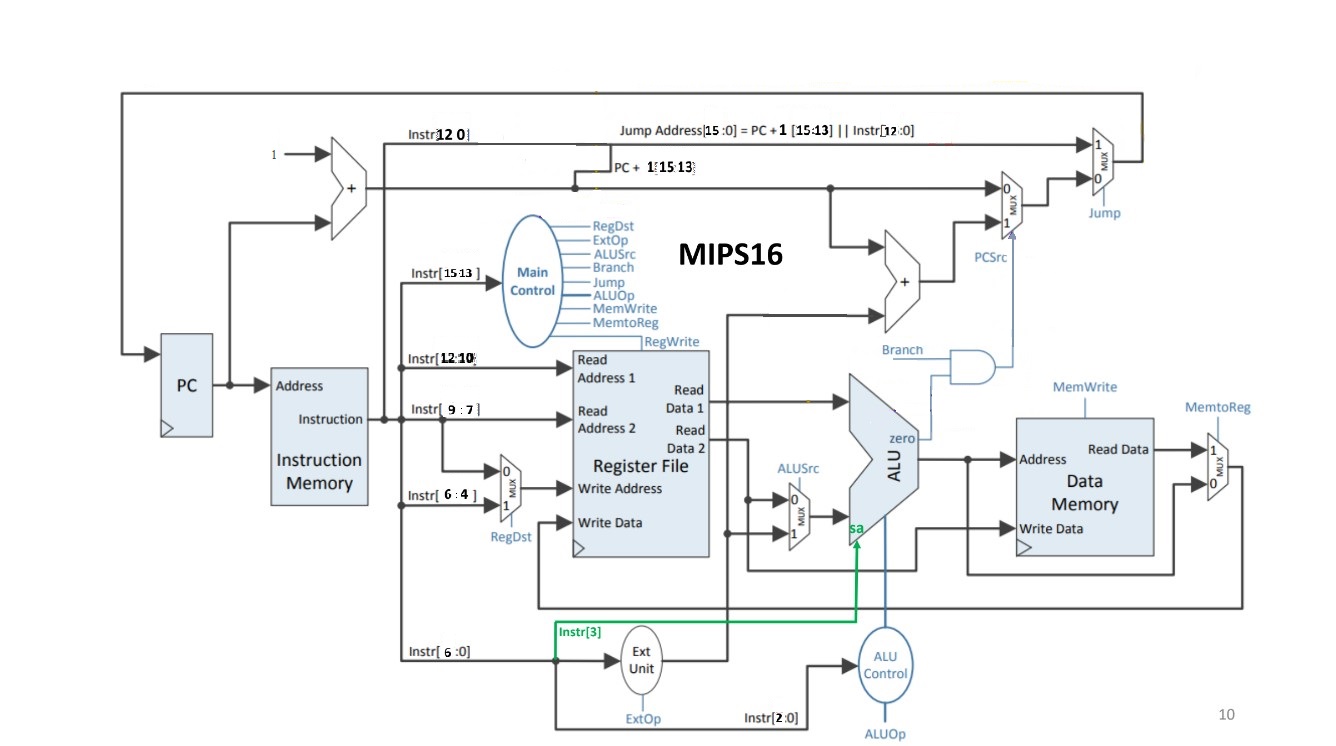
Ciclul unic înseamnă că microprocesorul execută fiecare instrucțiune într-un singur ciclu de ceas. Acest lucru înseamnă că instrucțiunile sunt procesate într-un timp constant, ceea ce poate fi mai rapid decât alte arhitecturi care folosesc mai mulți cicli pentru a executa o instrucțiune.

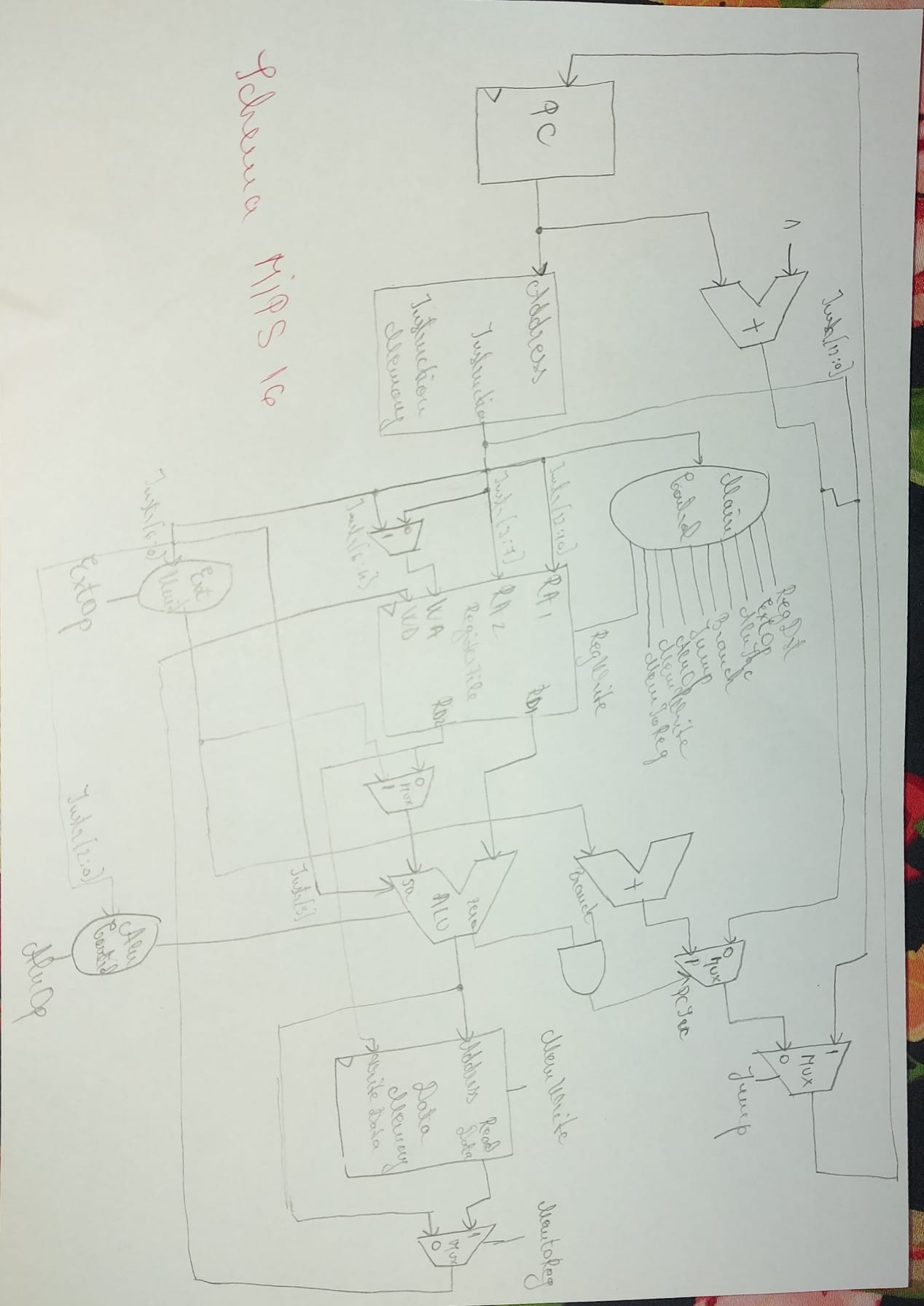
Pentru fiecare dintre aceste faze s-a creat o sursă separată în proiectul VHDL, iar componentele au fost incluse în main-ul *test\_env* (exceptând WB – care este implementată ca un multiplexor MUX 2:1). De asemenea, pentru a evita fenomenul de bounce și pentru a afișa rezultatele pe SSD-ul plăcii FPGA Basys 3, au fost implementate și incluse un MPG (Monopulse Generator), respectiv și un SSD (Seven Segment Display) în cadrul primelor laboratoare.

# **5. Scheme incluse**

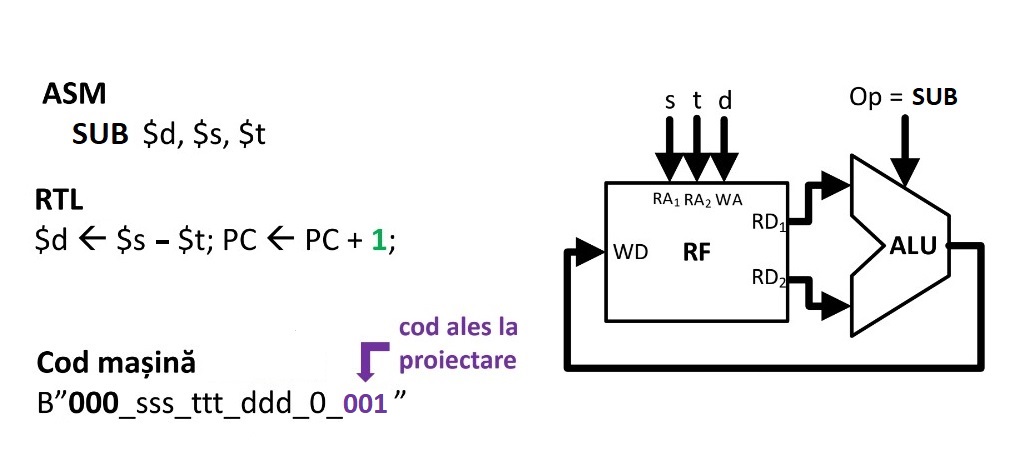
Voi include atât schemele preluate din <https://users.utcluj.ro/~vcristian/AC.html> , cât și cele întocmite de mine, pentru lizibilitate.

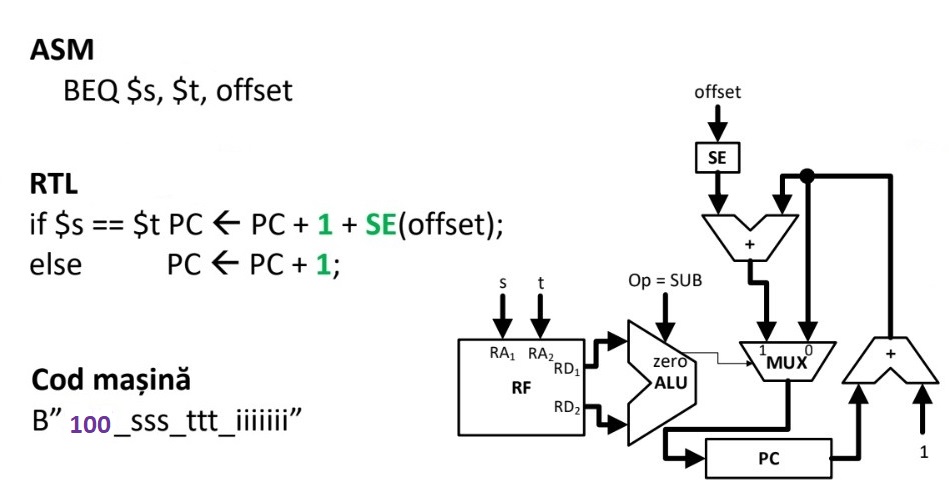
**Schema MIPS 16**

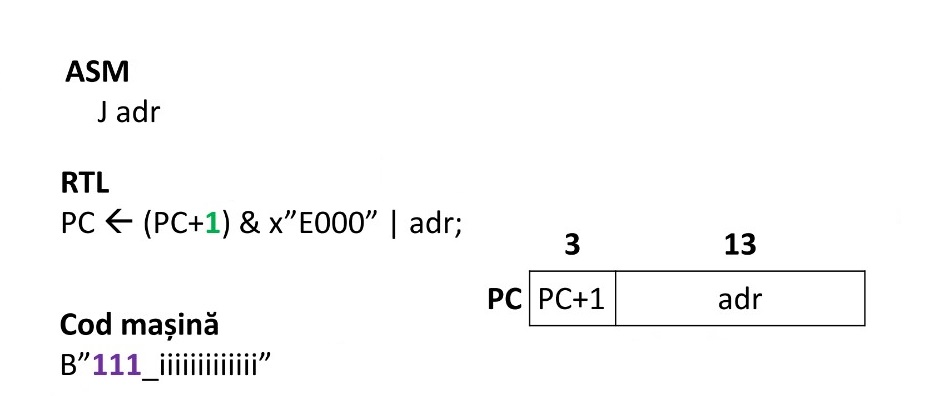
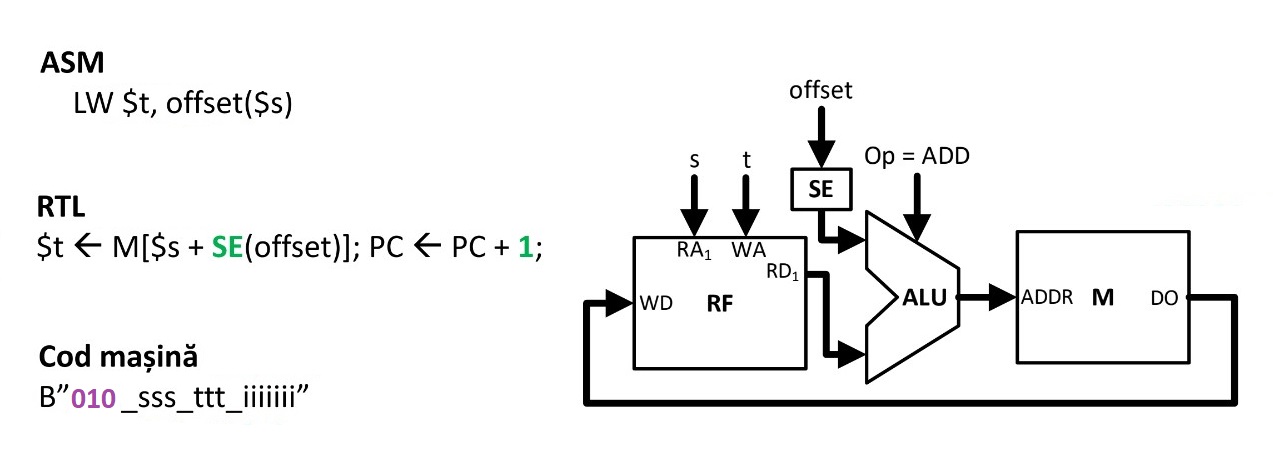
****

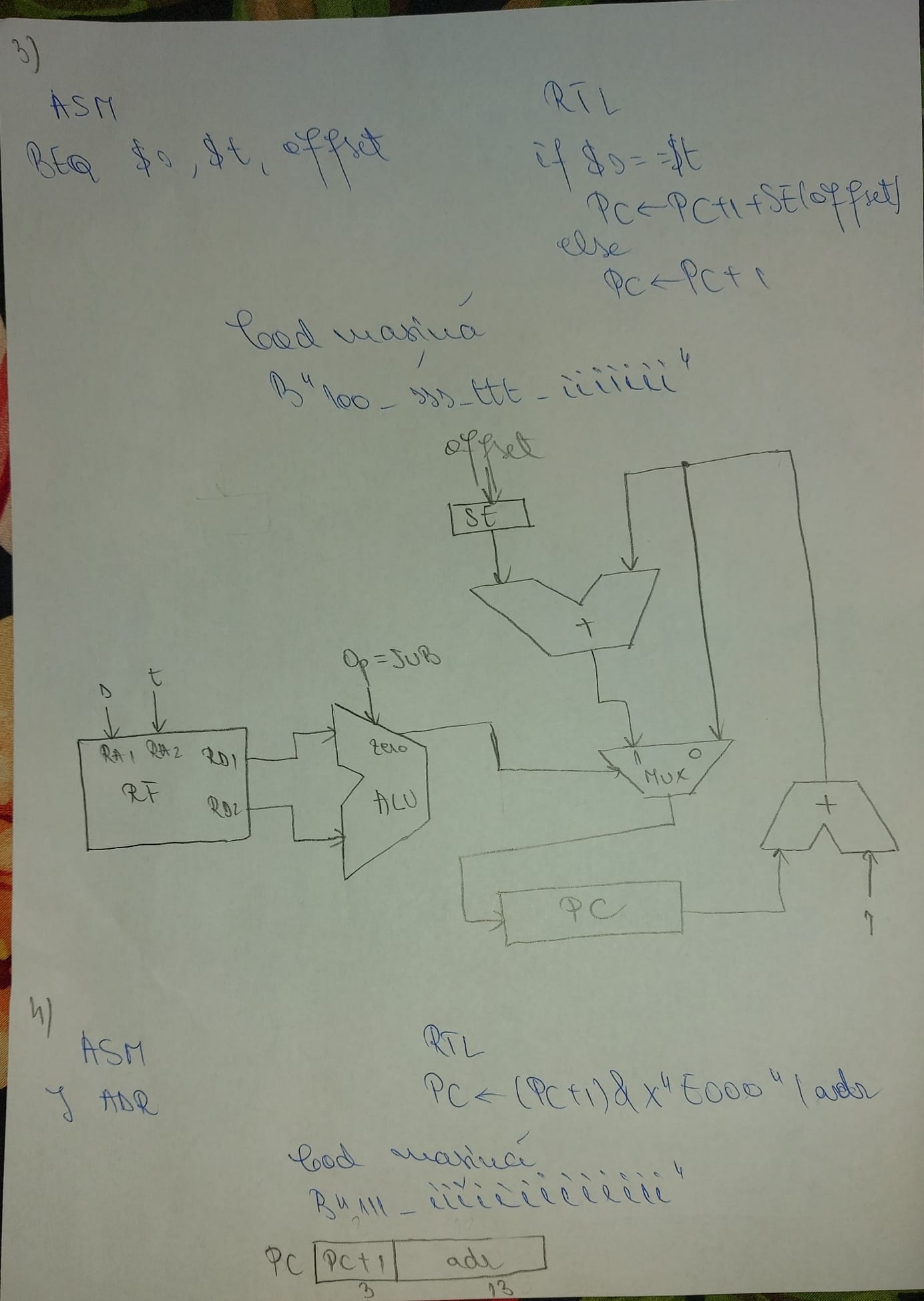
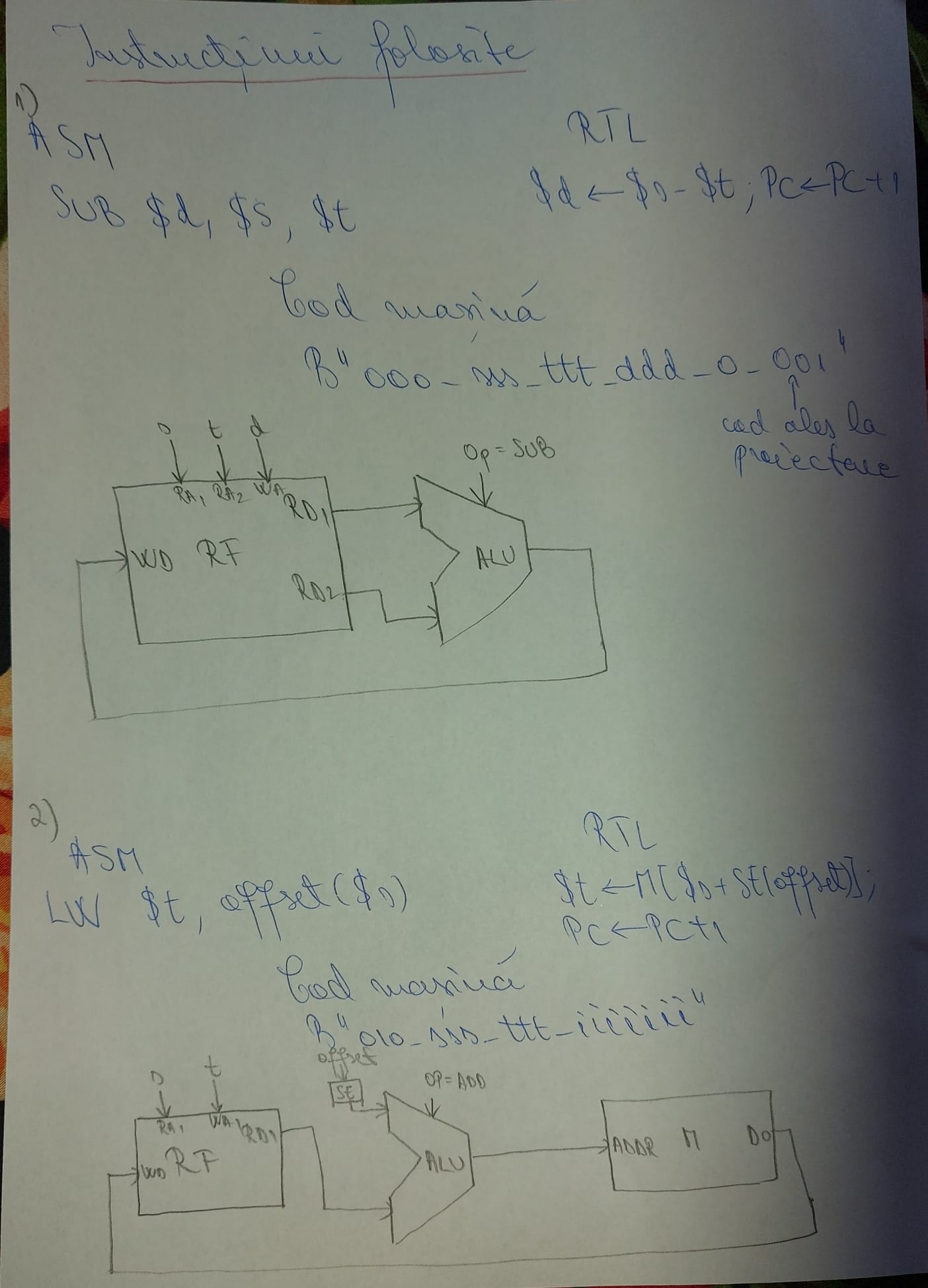
****

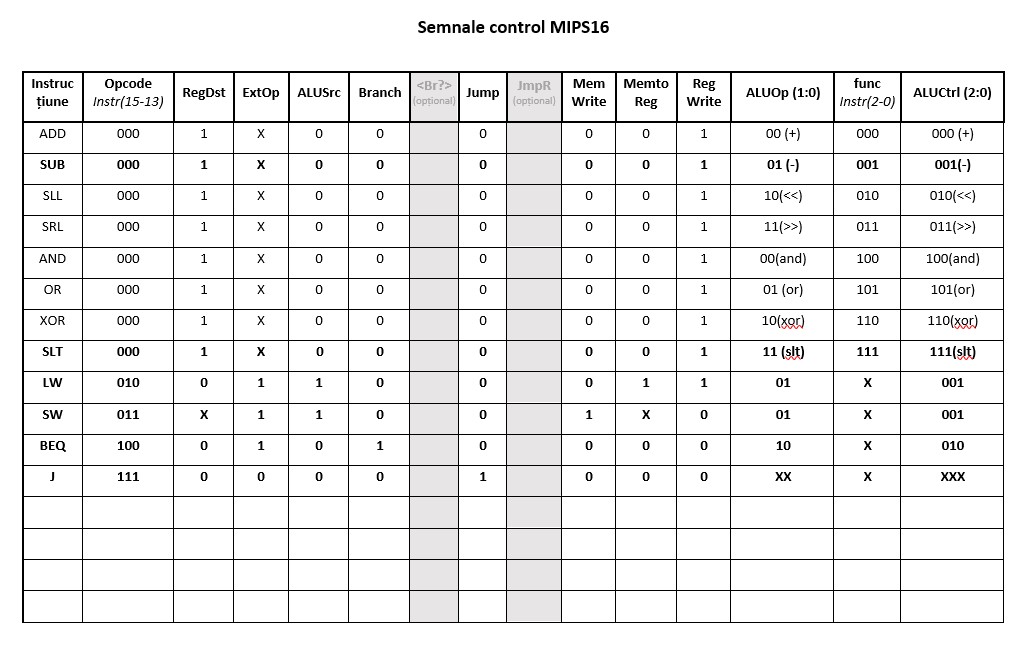
**Instrucțiuni folosite**

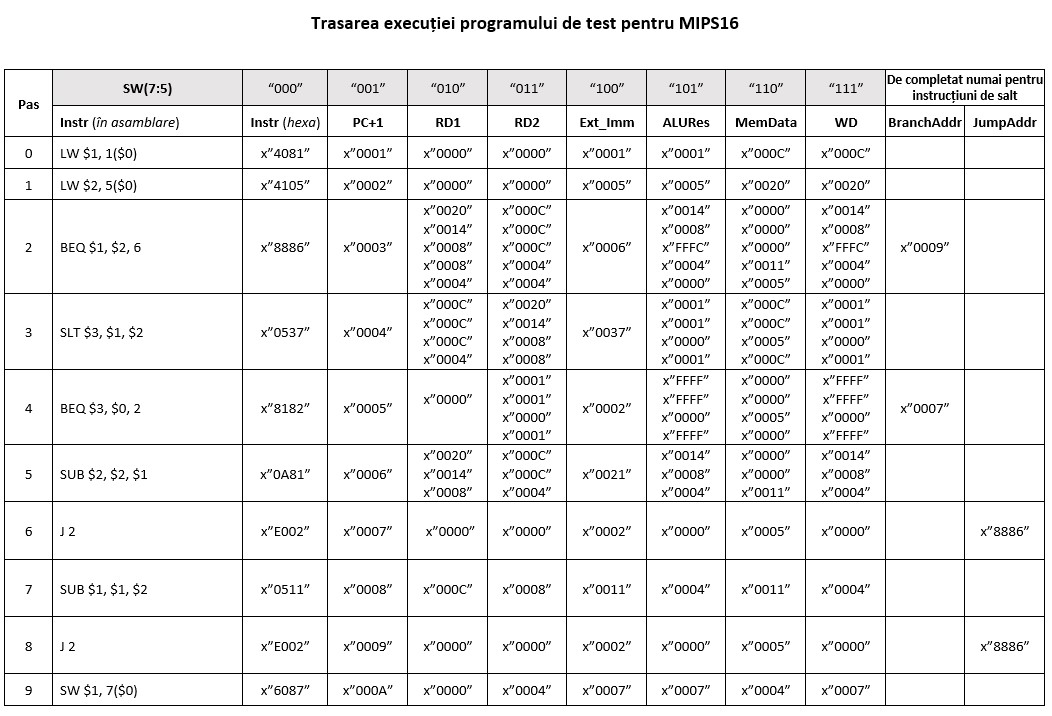
****

****

****







# **6. Concluzii**

Cunoștințele dobândite în cadrul laboratoarelor 4-7 la Arhitectura Calculatoarelor au condus la întocmirea microprocesorului MIPS 16 care realizează corect programul întocmit în cod mașină. Acesta a fost testat și verificat pe placa Basys 3 pentru corectitudine și certitudine.

# **7. Resurse necesare**

* Cunoștinte generale de hardware și programare dobândite la materiile Proiectare Logică, Proiectarea Sistemelor Numerice, Arhitectura Calculatoarelor, Programare în Limbaj de Asamblare
* Mediul de dezvoltare Xilinx VIVADO HL WebPACK (am folosit versiunea 2016.4)
* Basys 3 Artix-7 FPGA Trainer Board - Digilent

# **8. Bibliografie**

* <https://en.wikipedia.org/wiki/Stanford_MIPS>
* <https://users.utcluj.ro/~onigaf/files/teaching/AC/AC_indrumator_laborator.pdf>
* <https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/366-0.pdf>
* <https://users.utcluj.ro/~onigaf/files/AC.html>
* <https://users.utcluj.ro/~vcristian/AC.html>
* <https://www.youtube.com/@mirceapaulmuresan>